COMP 1010 - Introduction to Computer Science 1

Course Description

Calendar entry

(Lab Required) An introduction to computer programming using a procedural high-level language. May not be held with COMP 1011 or COMP 1012 or COMP 1013. Prerequisite: any grade 12 or 40S Mathematics, or equivalent.

General Course Description

This course is an introduction to implementing simple algorithms by writing computer programs. The course does not require previous programming experience. However, many students find that if they have never written any computer programs or have never been formally introduced to algorithms before, the pace can be challenging. If you would like to learn more about the subject of Computer Science, while also picking up important programming fundamentals, you may want to consider taking COMP 1500 *before* this course.

Detailed Prerequisites

Before entering this course, a student should be able to:

- Evaluate arithmetic expressions, applying the rules of order of operation to basic arithmetic operators (+, −, ×, ÷) and parentheses.
- Use common mathematical functions such as absolute value, square root, power functions, and trigonometric functions (e.g., to calculate angles).
- Download, install, and use new unfamiliar software on a desktop or laptop computer.

Course Goals

By the end of this course students will:

- Have a robust mental model for how the computer executes programming instructions, and makes execution flow decisions, by accessing information from memory, performing calculations, and storing results in memory.
- Write and run moderately complex programs using a procedural programming language.
- Devise solutions to simple problems and implement them as computer programs.
- Read and evaluate written programs.
- Describe basic programming concepts and structures in plain English.

- Represent ideas and information in a way that computers can understand and act on.
- Analyze and implement simple common algorithms for tasks such as searching.

Learning Outcomes

The Mechanics of Programming

Students should be able to:

1. Write and edit code in a text editor or simple IDE.
2. Compile and run their programs, providing interactive input and producing output.
3. Explain the conceptual differences between human-readable source code and object code/executables and the role of the compiler in translating from the former into the latter.
4. Find and correct errors that occur at compile time.
5. Determine the source of run-time errors using simple debugging techniques such as “trace” output statements.
6. Describe the step-by-step execution of a simple program without the use of a computer.
7. Apply programming standards, such as naming, commenting, and named constants, to produce human-readable and modifiable programs.

Data and Representation

Students should be able to:

1. Identify the data types of literal values.
2. Declare, initialize, and assign variables of primitive, string, and simple collection (array) types.
3. Apply operators and parentheses to build expressions using variables and literal values.
4. Determine the order of operations and the type of the result of an expression.
5. Use casting to change the type of a value.
6. Explain the consequences of numbers having limited ranges or precision when being represented in binary by a computer.
7. Describe the scope of variables and the uses of variables of different scope.
8. Identify the difference between references and objects for built-in types such as Strings and arrays.

Mathematical Operations

Students should be able to:

1. Write and evaluate arithmetic expressions (+, −, ×, ÷, mod) that include both literal values and variables.
2. Apply the modulo operator in common operations such as even/odd or “clock” math.
3. Use the compound assignment operators (like += or --) to simplify common assignment statements.
4. Use language libraries for evaluating common math functions.

Boolean Operations

Students should be able to:

1. Use relational operators on primitive types to produce Boolean results.
2. Evaluate Boolean expressions with logical operators (e.g., and, or, not).
3. Recognize the correspondence between Boolean conditions and Boolean variables.

String Processing

Students should be able to:

1. Declare and use string variables in a program.
2. Access characters in a string by index and build strings character-by-character.
3. Perform operations on individual characters such as case conversion.
4. Perform simple operations on strings such as concatenation.
5. Convert between numeric and string types.
6. Compare strings for equality.

Conditional Statements (if-else)

Students should be able to:

1. Write code that uses if and if-else constructions for decision making.
2. Write code that uses nested if statements and if-else-if chains for making more complex decisions.

Loops

Students should be able to:

1. Write code that uses deterministic for loops, and non-deterministic loops with while.
2. Convert between the two types of loops, while recognizing which is more appropriate for a given context.
3. Write code that uses nested loops, and other nested control structures.

Methods or Functions

Students should be able to:

1. Subdivide complex problems into subroutines.
2. Implement subroutines with parameters and return values.
3. Identify the scope of local variables.

Arrays

Students should be able to:

1. Determine when an array is an appropriate solution to a problem.
2. Declare and use one-dimensional arrays of primitive and string types.
3. Implement parallel arrays to store compound data types.
4. Implement partially filled arrays for data sets of an unpredictable size.
5. Pass arrays to and return arrays from subroutines.
6. Explain the difference between a reference to an array and an array object.
7. Describe how arrays are passed to subroutines as variable parameters.

Algorithms

Students should be able to:

1. Implement simple algorithms in a high-level programming language.
2. Devise algorithms to solve simple problems, such as array comparison or finding a minimum value.
3. Describe the linear and binary search algorithms, recognizing where each is appropriate.
4. Implement the linear search algorithm.